Final Exam

Date: May 9, 2018

UT EID: _____ _____

Printed Name: ___King _____Solver_____
Last, First

Your signature is your promise that you have not cheated and will not cheat on this exam, nor will you help others to cheat on this exam:

Signature: _____

Instructions:

- Closed book and closed notes. No books, no papers, no data sheets (other than the last two pages of this Exam)
- No devices other than pencil, pen, eraser (no calculators, no electronic devices), please turn cell phones off.
- Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space (boxes) provided. Do Not write answers on back of pages
- You have 180 minutes, so allocate your time accordingly.
- Unless otherwise stated, make all I/O accesses friendly.
- Please read the entire exam before starting.

Problem 1	20	
Problem 2	10	
Problem 3	10	
Problem 4	10	
Problem 5	15	
Problem 6	20	
Problem 7	15	
Total	100	

[20 points] Problem 1: Fundamentals. Answer the following short questions in the boxes provided.

(i) (**2pt**) In what region of memory on the microcontroller is the space for **local variables** allocated?

STACK

(ii) (**2pt**) What is the **degree of intrusiveness** of the debugging instrumentation you implemented in Lab 4: *Non-intrusive*, *Minimally intrusive* or, *Highly intrusive*?

Minimally Intrusive

(iii) (**3pt**). Assume the bus clock frequency is 16MHz. Assuming $10^6 \approx 2^{20}$. What is the maximum time between periodic interrupts for which one can setup using the **SysTick timer**?

$2^{24}/(16 \times 10^6) = 2^{24}/(2^4 \times 2^{20})$ = 1 second

(iv) (**3pt**) Assume the bus clock frequency is 80MHz. Two TM4C123 microcontrollers are communicating using **UART**. How many total bits are transmitted when sending the message: Hello_mBed

10 bytes with 2 bits per byte extra for Start+Stop = 10x10 = 100 bits

(v) (**2pt**) **Which registers are pushed** on the stack by hardware when an interrupt is serviced?

8 Registers: R0-R3, R12, LR,PC, PSR

(vi) (**4pt**) Which of the following are improper things to do inside a **periodic SysTick interrupt** service routine? (enter letters a-f in the box for all that apply)

a. Turning off SysTick interrupts
b. Communicating data to the main thread using a register

b, d

c. Writing conditional statements
d. Writing busy-wait loops
e. Using the stack by pushing 10 words then popping 10 words
f. Execute BX LR

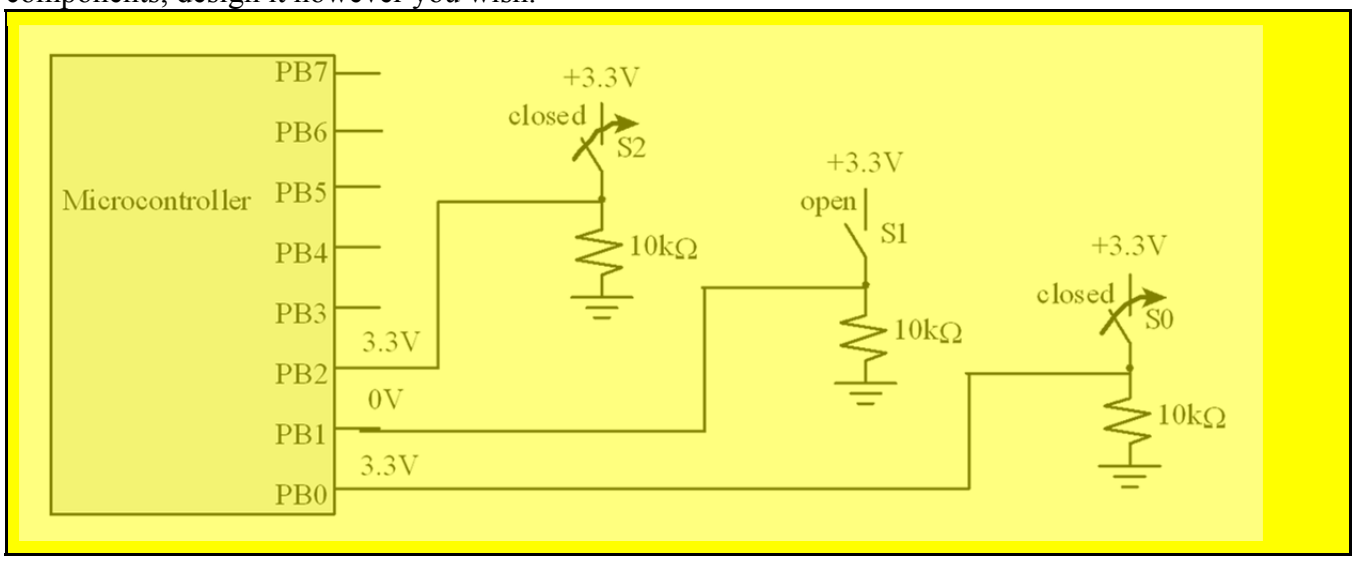
(vii) (**4pt**) Assume the bus clock is 16 MHz. Given the following UART baud-rate initialization, what is the actual **baud rate** in bits-per-sec?

Divisor = 2+ 32/64= 2.5
BR = $16 \times 10^6/(16 \times 2.5)$
= 400 kbps

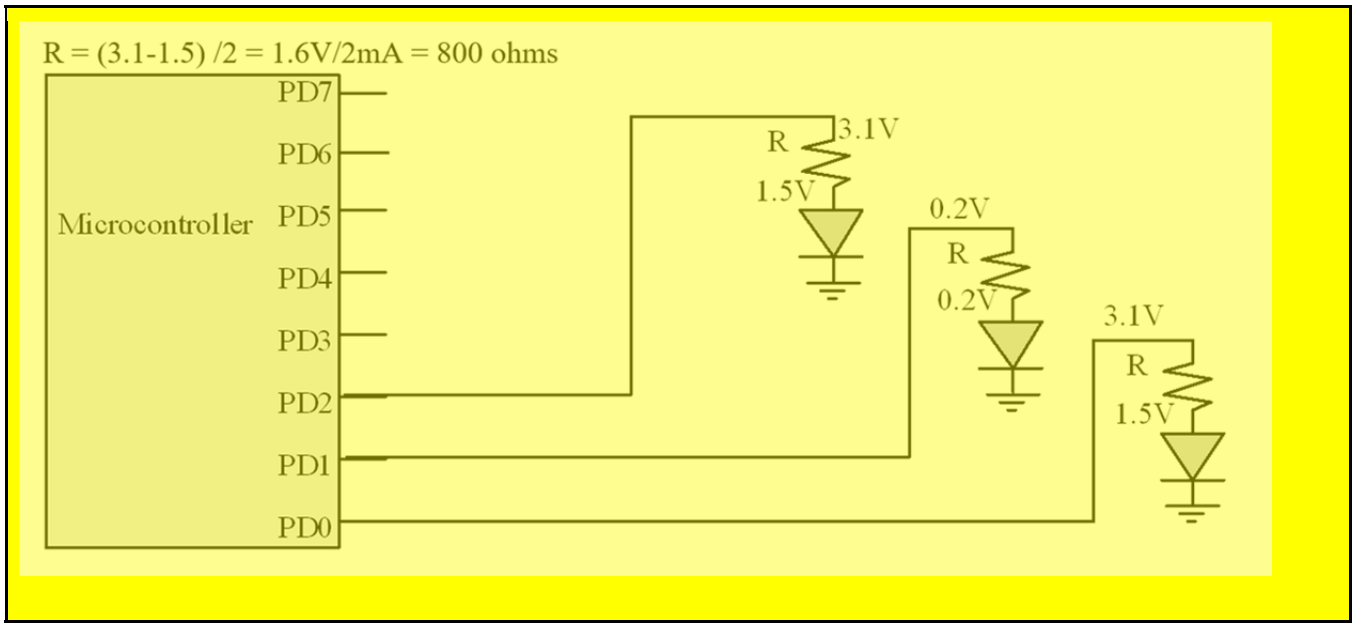
```
UART0_IBRD_R = 2;
UART0_FBRD_R = 32;
```

[10 points] Problem 2: **Circuits**.

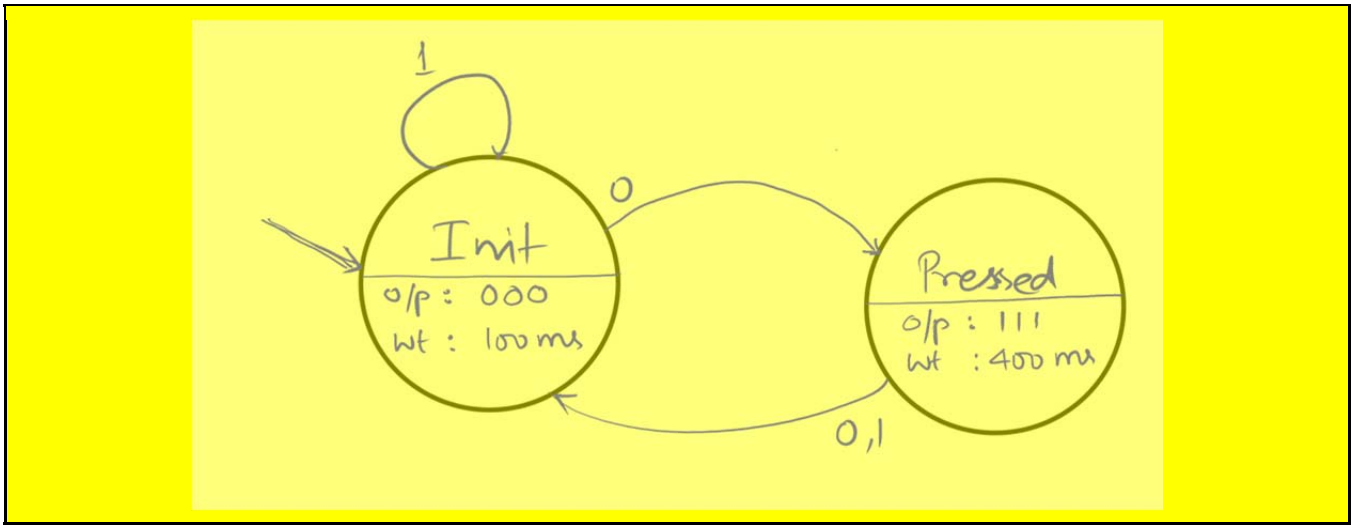
(5) Part a) Design a circuit that allows the user to input an integer value from 0 to 7 (indicated by the switches). You must interface multiple switches and resistors like those used in lab. You can use any pin available on Port B. Port B is initialized to input, without internal pull up or pull down. Label on your circuit all voltages occurring when the user wishes to input the value 5. Show connections to +5V, +3.3V and ground as needed. Specify resistor values as needed. It is not necessary to minimize components, design it however you wish.



(5) Part b) Design a circuit that allows the user to output a value from 0 to 7 (indicated by the LEDs). You must interface multiple red LEDs (V_d = 1.5V, I_d = 2mA) using positive logic. Show resistor values with calculations. You can use any pin available on Port D. Assume Port D is initialized to output. Label on your circuit all voltages occurring when the user wishes to output the value 5. Assume the output low voltage of the TM4C123 is 0.2V. Assume the output high voltage of the TM4C123 is 3.1V. Assume the output low voltage of the 7406 is 0.5V. Show connections to +5V, +3.3V and ground as needed. Specify resistor values as needed. It is not necessary to minimize components, design it however you wish.



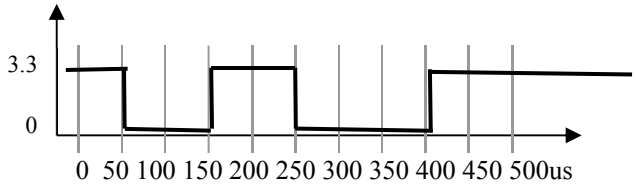
[10 points] Problem 3: Finite State Machine. Design a FSM that implements a 2Hz flashing LED with a 80% duty-cycle. The system takes a single input on button PF4, and has 3 LED outputs PF3(*Green*), PF2(*Blue*) and PF1(*Red*). When the button (interfaced using negative logic) is pressed, the LEDs flash a *White* color at 2Hz, 80% duty cycle. When the button is not pressed all LEDs are Off. **Draw a Moore State Transition Graph**(STG) for the FSM, clearly showing the states, transitions, dwell times, inputs and outputs. For full credit, minimize the number of states in your STG.



[10 points] Problem 4: **UART.**

(5) Part a) Assuming this is one 10-bit frame, think about which is the start bit, 8 data bits, and stop bit. Assume the line is idle before and after this one transmission. What is the 8-bit data?

Data = 11000110 = 0xC6



(3) Part b) What is the baud rate of this channel in bits/sec?

BR = 1/bit-time = 1/50us = 10^6/50 = 20kbps

(2) Part c) In general, let **BR** be the baud-rate of a UART channel in bits/sec. What is the general relationship for **maximum bandwidth** as a function of **BR**?

Max BW = (No. of. Useful bits / Total number of bits) x BR

[15 points] Problem 5: **programming, design.**

Consider a situation where there can be up to 100 interconnected devices. Devices can send data to other devices. The communication links are not bidirectional. Each device can send data to *up to* ten other devices. However, there is no limit to the number of devices from which a device can receive data. For example if there were 100 devices, one device might be able to receive data from all the other 99. As a second example, consider this

situation with 5 devices such that 0 can send data to 3, 1 can send data to 0+3, 2 can send data to 3+4, 3 can send data to 1, and 4 cannot send any data at all.

Design a struct named device_t that contains two 8-bit unsigned integer fields battery and work plus other fields to record other devices to which this device can send data. A device's battery is the number of hours the device has to live and the work is the hours of work it has left to do.

Implement a function fail(...) that receives an array of all devices (of type device_t) and computes as output (not the return value) the array of failed devices, which are defined as those that have more work than battery. The hard part of this question is to design the function prototype that allows you to pass in an array of all devices and to return a populated array of failed devices.

Implement a function popular(...) that receives an array of all devices (of type device_t) and returns as output the index of the most popular device in the array -- i.e., the device that can receive data from the most other devices (device 3 is most popular in the above figure).

```
// Define your struct device_t here. (3 points)

#define MAX 100
struct device{
  uint8_t battery,work;
  uint8_t numConnections; // number of output connections
  uint8_t connect[10];     // index to other device
};
typedef struct device device_t;
```

```
// Define the function prototype for fail() here. (2 points)
 void fail(device_t world[MAX], // possible devices
            uint8_t numDevs,          // number of devices in the world
            device_t failed[MAX],
            uint8_t *numFailed  // pass by reference
    );
// Implement fail() here. (4 points)


void fail(device_t world[MAX], // possible devices
            uint8_t numDevs,          // number of devices in the world
            device_t failed[MAX],
            uint8_t *numFailed){ // pass by reference
  *numFailed = 0;
  for(int i=0;i<NumDevs;i++){
            if(world[i].battery < world[i].work){ // failed?
            failed[*numFailed] = world[i]; // populate return array
            (*numFailed)++;
            }
  }
}
```

```
// Define and implement popular() here. 6 points.
```

[20 points] Problem 6: **ADC and DAC.**

You are given a transducer that converts *distance* to voltage (*V1*). You may assume the *distance* signal contains frequency components from 0 to 1000 Hz. You will build an embedded system with an ADC to sample *V1*, software to perform calculations, and a DAC to output the analog voltage *V2*.

The transducer has a monotonic transfer function shown on the left. The goal of the system is to create an overall linear response as shown on the right. To create this response, your embedded system will implement the transfer function in the middle. Assume both the ADC and the DAC have an 8-bit precision and a range of 0 to +3.3V. Let *N* be the 8-bit integer returned by the ADC, and *M* is the 8-bit integer needed to be output to DAC.

In equation form, the transducer response is

V1 = (0.825V/4cm)**distance* if $0 \le$ *distance* < 4 cm,
V1 = (2.475V/4cm)**distance* – 1.65V if $4 \le$ *distance* < 8 cm,

In equation form, your system response should be (*note*: 85.333 ≈ 256/3)

M = 2**N* if 0 ≤ *V1* < 0.825 V,
M = (0.667)**N*+85.333 if 0.825V ≤ *V1* < 3.3V,

In equation form, the overall desired response should be linear from distance to V2

V2 = (3.3V/8cm)**distance*

You may not use floating point; please implement the software using fixed point.

You are given ADC and DAC software functions, with prototypes

```
uint8_t ADC_In(void);  // sample ADC
void DAC_Out(uint8_t); // output to DAC
```

The SysTick interrupt service routine will periodically execute

```
void SysTick_Handler(void){ uint8_t N,M;
  N = ADC_In();    // given to you
  M = Convert(N);  // you will write this in C
  DAC_Out(M);      // given to you
}
```

(5) Part a) Fill in the following table describing the operation of the ADC.

Distance(cm)	*V1* (V)	**N=ADC_In()** (8-bit)

0	0.0	0
2	0.4125	32
4	0.825	64
5.333	1.650	128
6.667	2.475	192

(10) Part b) Write a C function **`Convert`** that converts the ADC input to the DAC output. Do not implement a table lookup; rather, implement piece-wise linear response from 8-bit input to 8-bit output.

```
uint8_t Convert(uint8_t n){uint32_t m; // promotion
  if(n<64){ // this could be (n<=64)
     m = 2*n;
  }else{
     m = (2*n+256)/3;          // M = (0.667)*N+85.333
     m = (667*n+85333)/1000; // M = (0.667)*N+85.333
     m = (683*n+87381)>>10;  // M = (0.667)*N+85.333
  }
  return m; // m is bounded from 0 to 255, so no overflow
}
```

(5) Part c) Write a C function to initialize SysTick interrupts as appropriate for this problem. Set the priority to 0, so it has highest priority. Please arm and enable interrupts. Assume bus clock is 80 MHz.

```
// interrupt faster than 2000 Hz, 500us,
void SysTick_Init(uint32_t clock){long sr;
  NVIC_ST_RELOAD_R = 39999; // any reasonable value smaller ok
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF); // priority
  NVIC_ST_CTRL_R = 0x07;



  Enable_Interrupts();
}
```

[15 points] Problem 7: **Local Variables.**

You will implement the Function nCr (n choose r). It takes two input parameters, n and r and returns the number of r-element combinations that can be formed using n elements. n and r are both unsigned 8-bit integers. Your function will return a 32-bit unsigned value. Note that 0!=1. Parameter passing follows AAPCS. Do not try to optimize, implement the formula and, no need to account for any overflow conditions.

uint32_t nCr(uint8_t n, uint8_t r); // n,r <= 10;

(8) Part a) Write nCr in assembly using 3 local variables (nFact, rFact, nrFact) with binding to store the intermediate results nFact=n! rFact=r! and nrFact=(n-r)! You are given a Factorial function implemented in C, which you can call: uint32_t Factorial(uint8_t k); // factorial of k

```
; Input: n, r
; Output: n choose r
nFact EQU 0   ; Binding
rFact EQU 4
nrFact EQU 8
nCr
   PUSH {R4,LR}
   SUB SP, #12  ; Allocation
   MOV R4,R0
   BL Factorial   ; R0 has n
   STR R0,[SP,#nFact] ; nFact has n!
   MOV R0,R1      ; R0 has k
   BL Factorial
   STR R0,[SP,#rFact] ; rFact has r!
   SUB R0,R4,R1
   BL Factorial
   STR R0,[SP,#nrFact] ; rFact has (n-r)!
   LDR R1,[SP,#rFact] ; R1 has r!
   LDR R4,[SP,#nrFact] ; R4 has (n-r)!
   MUL R1,R4
   LDR R0,[SP,#nFact] ; R0 has n!
   UDIV R0,R1
   ADD SP,#12  ; De-Allocation
   POP {R4,LR}
  BX LR
```

BX LR

(4) Part b) Assuming the following call is made: nCr(5,2), show the contents of the stack (and the SP) just before the local variables are de-allocated. Assume SP is 0x20000ABC initially.

```
SP(0x20000AA8) ->  120
  (0x20000AAC)     2
  (0x20000AB0)     6
  (0x20000AB4)     old R4
  (0x20000AB8)     old LR
  (0x20000ABC)     xxxxxx
```

(3) Part c) Show an assembly call to the subroutine nCr to compute 10 choose 3.

```
MOV R0,#10
MOV R1,#3
BL  nCr
```